Control Flow Graph
Announcements

- ECE colloquium on Monday 9/11 from 4:30pm, PHL 233
  - “Smart Healthcare” by Prof. Niraj Jha of Princeton

- Lab 1 is due tomorrow @11:59pm

- First problem set (HW1) will be released tomorrow

- Sign up for the first student-led discussion (Tue 9/26)
  - Three papers posted on the course website
    - [https://www.csl.cornell.edu/courses/ece5775/schedule.html#seminar1](https://www.csl.cornell.edu/courses/ece5775/schedule.html#seminar1)
  - 3 students per talk, 3 talks per session
  - Bidding starts tomorrow @5pm on Piazza
    - First come first serve
Revisiting Some Important Graph Definitions

- What is a strongly connected component?
- What is a directed tree?
- Is a directed tree strongly connected?
- Is a directed tree also a DAG?
- Is a DAG strongly connected?
Agenda

- More on BDDs and static timing analysis

- Basics of control data flow graph
  - Basic blocks
  - Control flow graph

- Dominance relation
  - Finding loops
Name the logic gates represented by the following BDDs

(a) 

(b)
BDD Application on Functional Verification

- Either prove equivalence
- Or find counterexample(s)
  - Input values \((x, y)\) for which these programs produce different results

```c
bool P(bool x, bool y) { return ~(~x & ~y); }

bool Q(bool x, bool y) { return x ^ y; }
```

Straight-line evaluation

| v1 = ~x |
| v2 = ~y |
| v3 = v1 & v2 |
| P = ~v3 |
| Q = x ^ y |

Is \((P == Q)\) true?
BDD-based Equivalence Checking

\[ P = \neg v3 = \neg(\neg x \& \neg y) = x \lor y \]

\[ Q = x \xor y \]

\[ T = (P \equiv Q) \]

Counterexample:
Setting \( x = 1 \) & \( y = 1 \) leads to false output

Hence \( P \) does not equal \( Q \)
FPGA LUT Mapping Revisited

- Cone $C_v$: a subgraph rooted on a node $v$
  - K-feasible cone: $\#\text{inputs}(C_v) \leq K$ (Can occupy a K-input LUT)
  - K-feasible cut: The set of input nodes of a K-feasible $C_v$

Another 3-feasible cone with an associated cut $= \{a, b, c\}$

A 3-feasible cone with a cut $= \{c, e, f\}$
Timing Analysis with LUT Mapping

- **Assumptions**
  - $K=3$
  - All inputs arrive at time 0
  - Unit delay model: 3-LUT delay = 1; Zero edge delay

- **Question:** Minimum arrival time (AT) of each gate output?

AT(a) = 1, AT(d) = 1, AT(b) = 1, AT(e) = 1, AT(c) = 1, AT(f) = ?, AT(g) = ?, AT(h) = ?
FPGA Design Flow with HLS

High-level Programming Languages

- Compilation
- Scheduling/Pipelining, Binding

RTL

- Logic Synth., Tech. Mapping, P&R, STA

Bitstream

HLS Flow

RTL Flow

if (condition) {
  ...
} else {
  t_1 = a + b;
  t_2 = c \ast d;
  t_3 = e + f;
  t_4 = t_1 \ast t_2;
  z = t_4 - t_3;
}

Timed design

Untimed high-level description

FPGA
A Typical HLS Flow

- **High-level Programming Languages**: (C/C++, OpenCL, SystemC, ...)
- **Parsing**
- **Transformations**
- **Intermediate Representation (IR)**
- **Allocation**
- **Scheduling**
- **Binding**
- **RTL generation**

### Code Example

```c
if (condition) {
    ...
} else {
    t_1 = a + b;
    t_2 = c * d;
    t_3 = e + f;
    t_4 = t_1 * t_2;
    z = t_4 - t_3;
}
```

### Control Data Flow Graph (CDFG)

### Finite State Machines with Datapath

3 cycles
Intermediate Representation (IR)

- Purposes of creating and operating on an IR
  - Encode the behavior of the program
  - Facilitate analysis
  - Facilitate optimization
  - Facilitate retargeting

- The IR we will focus on is control data flow graph (CDFG)
Program Flow Analysis

- Control flow analysis: determine control structure of a program and build control flow graphs (CFGs)

- Data flow analysis: determine the flow of data values and build data flow graphs (DFGs)
Basic Blocks

- **Basic block**: a sequence of consecutive intermediate language statements in which flow of control can only enter at the beginning and leave at the end
  - Only the last statement of a basic block can be a branch statement and only the first statement of a basic block can be a target of a branch
Basic Block Partitioning Algorithm

- Identify leader statements (i.e. the first statements of basic blocks) by using the following rules:
  - (i) The first statement in the program is a leader
  - (ii) Any statement that is the target of a branch statement is a leader (for most intermediate languages these are statements with an associated label)
  - (iii) Any statement that immediately follows a branch or return statement is a leader
Example: Forming the Basic Blocks

(1) p = 0
(2) i = 1
(3) t1 = 4 * i
(4) t2 = a[t1]
(5) t3 = 4 * i
(6) t4 = b[t3]
(7) t5 = t2 * t4
(8) t6 = p + t5
(9) p = t6
(10) t7 = i + 1
(11) i = t7
(12) if i <= 20 goto (3)
(13) j = ...

Basic Blocks:

B1
(1) p = 0
(2) i = 1

B2
(3) t1 = 4 * i
(4) t2 = a[t1]
(5) t3 = 4 * i
(6) t4 = b[t3]
(7) t5 = t2 * t4
(8) t6 = p + t5
(9) p = t6
(10) t7 = i + 1
(11) i = t7
(12) if i <= 20 goto (3)

B3
(13) j = ...

Leader statement is:
(1) the first in the program
(2) any that is the target of a branch
(3) any that immediately follows a branch
Control Flow Graph (CFG)

- A control flow graph (CFG), or simply a flow graph, is a directed graph in which:
  - (i) the nodes are basic blocks; and
  - (ii) the edges are induced from the possible flow of the program.

- The basic block whose leader is the first intermediate language statement is called the entry node.

- In a CFG we assume no information about data values:
  - an edge in the CFG means that the program may take that path.
Example: Control Flow Graph Formation

(1) \( p = 0 \)
(2) \( i = 1 \)
(3) \( t1 = 4 \times i \)
(4) \( t2 = a[t1] \)
(5) \( t3 = 4 \times i \)
(6) \( t4 = b[t3] \)
(7) \( t5 = t2 \times t4 \)
(8) \( t6 = p + t5 \)
(9) \( p = t6 \)
(10) \( t7 = i + 1 \)
(11) \( i = t7 \)
(12) if \( i \leq 20 \) goto (3)
(13) \( j = \ldots \)
Dominators

- A node $p$ in a CFG **dominates** a node $q$ if every path from the entry node to $q$ goes through $p$. We say that node $p$ is a **dominator** of node $q$

- The **dominator set** of node $q$, $\text{DOM}(q)$, is formed by all nodes that dominate $q$
  - By definition, each node dominates itself therefore, $q \in \text{DOM}(q)$
Definition: Let $G = (N, E, s)$ denote a CFG, where

$N$: set of nodes

$E$: set of edges

$s$: entry node and

let $p \in N$, $q \in N$

- $p$ dominates $q$, written $p \leq q$
  - $p \in \text{DOM}(q)$

- $p$ properly (strictly) dominates $q$, written $p < q$ if $p \leq q$ and $p \neq q$

- $p$ immediately (or directly) dominates $q$, written $p <_{d} q$ if $p < q$
  and there is no $t \in N$ such that $p < t < q$
  - $p = \text{IDOM}(q)$
Example: Dominance Relation

- **Dominator sets:**
  \[
  \begin{align*}
  \text{DOM}(1) & = \{1\} \\
  \text{DOM}(2) & = \{1, 2\} \\
  \text{DOM}(3) & = \{1, 2, 3\} \\
  \text{DOM}(10) & = \{1, 2, 10\}
  \end{align*}
  \]

- **Immediate domination:**
  \[
  1 <_d 2, \quad 2 <_d 3, \quad \ldots
  \]
  \[
  \text{IDOM}(2) = 1, \quad \text{IDOM}(3) = 2 \quad \ldots
  \]
Dominance Question

Assume that node $P$ is an immediate dominator of node $Q$

Question: Is $P$ necessarily an immediate predecessor of $Q$ in the CFG?

Answer: **NO**
Identifying Loops

- **Motivation**: Programs spend most of the execution time in loops, therefore there is a larger payoff for optimizations that exploit loop structure.

- **Goal**: Identify loops in a CFG, not sensitive to input syntax.
  - Create an uniform treatment for program loops written using different syntactical constructs (e.g. while, for, goto).

- **Approach**: Use a general approach based on analyzing graph-theoretical properties of the CFG.
Definition: A **strongly connected component** (SCC) of the CFG, with
- a single entry point called the **header** which dominates all nodes in the SCC
Question: In the CFG below, nodes 2 and 3 form an SCC; but **do they form a loop?**
Finding Loops

Loop identification algorithm
- Find an edge $B \rightarrow H$ where $H$ dominates $B$; This edge is called a **back-edge**
  - Find all nodes that can reach $B$ only through nodes dominated by $H$; add them to the loop
    - $H$ and $B$ are naturally included
Finding Loops (1)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)
Finding Loops (1)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
Finding Loops (2)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7)
Intuition of Dominance Relation

Imagine a source of light at the entry node, and that the edges are optical fibers.

To find which nodes are dominated by a given node, place an opaque barrier at that node and observe which nodes become dark.
Finding Loops (2)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)   Entire graph
(10,7)
Finding Loops (2)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1)  Entire graph
(10,7)  {7,8,10}
Finding Loops (3)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7) \{7,8,10\}
(7,4)
Finding Loops (3)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7) {7,8,10}
(7,4)
Finding Loops (3)

Find all back edges in this graph and the natural loop associated with each back edge

(9,1) Entire graph
(10,7) {7,8,10}
(7,4) {4,5,6,7,8,10}
Summary

- **Basic Blocks**
  - Group of statements that execute atomically

- **Control Flow Graphs**
  - Model the control dependences between basic blocks

- **Dominance relations**
  - Shows control dependences between BBs
  - Used to determine natural loops
Before Next Class

- Sign up for paper presentations on Piazza
- Next lecture: SSA form and LLVM
Acknowledgements

- These slides contain/adapt materials developed by
  - Forrest Brewer (UCSB)
  - Ryan Kastner (UCSD)
  - Prof. José Amaral (Alberta)